**REPORT**

< 보물 찾기 >

슈가슈가룬

|  |  |
| --- | --- |
|  |  |
| 과 목 | C프로그래밍 |
| 담당 교수 | 박소영 |
| 제출 일자 | 2024. 06. 09. |
| 학과 | 컴퓨터공학부 |
| 학번 | 202311309, 202411397, 202411328, 202411236 |
| 이름 | 송은서, 홍지아, 유지혁, 강민재 |

- 목 차 -

Ⅰ. 과제 주요 기능 …………………………………………………………… p.1

Ⅱ. 주요 기능 구현을 위한 기술적 상세 명세 …………………………… p.3

Ⅲ. 토의 사항 …………………………………………………………………… p.13

Ⅳ. 기여도 ……………………………………………………………………… p.15

**Ⅰ. 과제 주요 기능**

**게임 환경 설정**

* 변수 초기화: 게임 변수를 초기화한다.
* 맵 초기화: 벽과 빈 공간의 위치를 초기화하고, 게임 내 오브젝트(플레이어, 몬스터, 보물 등) 위치를 랜덤으로 초기화한다.

**메뉴**

* 메인 메뉴: Game Start, Instructions, EXIT의 옵션을 제공한다. Instructions 선택 시 규칙을 보여준 후 다시 메인 메뉴로 돌아가고, Game Start을 누르면 난이도 선택한 후 게임을 시작한다. EXIT 을 누르면 종료되었다는 메시지를 출력 후 프로그램을 종료시킨다.
* 난이도 메뉴: 초급, 중급, 고급 메뉴를 선택할 수 있다.

**객체 이동**

* 플레이어 이동: 방향키를 이용하여 게임 플레이어를 이동시킬 수 있다.
* 몬스터 이동: 랜덤으로 몬스터가 이동한다.

**깃발과 장애물**

* 보물 찾기: 플레이어가 보물을 찾으면 메시지가 출력되고, 점수를 계산할지 묻는다.
* 이득 깃발: 3개의 깃발은 질문에 맞는 답을 잘 선택시 호감도(점수) 증가, 2개의 깃발은 보물 방향 힌트(각자 좌우, 상하 방향), 자동으로 호감도(점수) 증가의 효과를 가진다.  
  일정 확률로 랜덤 상점이 나오는데, 시간을 이용해서 플레이어의 속도 증가나 몬스터의 속도 감소 아이템을 얻을 수 있다. 또는 시간을 더 얻을 수 있다.
* 벌칙 깃발: 랜덤으로 플레이어의 속도 감소, 몬스터의 속도 증가 중 하나의 효과가 발동한다.
* 장애물: 랜덤으로 이동하며 플레이어와 접촉시 호감도가 100점씩 감소한다.

**난이도 설정**

* 게임 시작 전 메뉴에서 난이도를 선택할 수 있으며, 선택한 난이도에 따라 몬스터의 개수와 얻어야 하는 점수가 달라진다. 사용자가 선택한 난이도의 게임을 성공적으로 수행하면, 다음 난이도로 자동 이동한다.

**게임 진행**

* 제한 시간이 지나면 게임이 종료된다.
* 게임 중간에 esc 키 입력 시 메뉴로 돌아간다.
* 보물을 찾은 후, 플레이어가 점수를 계산한다고 하면 게임을 종료하며 점수를 계산해서 보여준다. 점수가 난이도 별 기준 이상이라면, 레벨업 또는 게임 클리어를 하고, 이하라면 게임 오버 되어 게임을 종료한 후 메뉴로 돌아간다.

**Ⅱ. 주요 기능 구현을 위한 기술적 상세 명세**

**게임 환경 설정(초기화)**

* initBoard(): 이중 for문을 통해 배열 map의 모든 셀을 순회한다. 맵의 가장자리에 벽(WALL)을 배치하고, 나머지 공간은 빈 공간(SPACE)으로 설정한다. 플레이어, 보물, 이득 깃발, 벌칙 깃발, 몬스터를 initRandomPosition(Pointer\* object)를 이용해 보드의 빈 공간에 랜덤하게 배치한다.
  + initRandomPosition(Pointer\* object): 게임 요소를 보드의 랜덤 위치에 배치한다. 선택된 랜덤 위치가 빈 공간(SPACE)인지 확인하고, 빈 공간일 경우 해당 위치를 게임 요소의 위치로 설정한다.
* initGameVariables(): 플레이어의 속도, 게임 변수, 점수 등을 초기화한다.

**메뉴**

* **printSelectionBox(int x, int y)**: 선택 상자를 주어진 위치에 출력한다.
* **eraseSelectionBox(int x, int y)**: 선택 상자를 주어진 위치에서 지운다.
* **menu()**: 게임 변수를 초기화하고, 콘솔 화면을 지운 후 시작 음악을 재생한다. 게임 영역과 printMenu()로 메뉴를 출력한 후, 메뉴 선택 상자를 출력하고, 사용자가 메뉴를 선택할 수 있도록 switchMenu() 함수를 호출한다.
* **printMenu()**: 메인 메뉴를 출력한다. "Game Start", "Instructions", "EXIT" 옵션을 표시한다.
* **switchMenu()**:
  + 사용자의 키 입력을 받아 메뉴를 선택한다.
  + 방향키로 선택 상자를 이동하고, 엔터키를 누르면 선택된 메뉴를 실행한다.
  + 선택된 메뉴에 따라 runMenu(int menuNum) 함수를 호출한다.
* **runMenu(int menuNum)**:
  + 사용자가 선택한 메뉴를 실행한다.
  + 1번 메뉴는 difficultyMenu()를 호출하여 난이도 선택 메뉴로 이동한다.
  + 2번 메뉴는 gameRule()을 호출하여 게임 규칙을 출력한다.
    - slowPrint() : 문자열을 출력하는 속도와 출력할 문자열을 인수로 받는다.

while(\*s)를 통해 엔터키가 문자열이 끝날 때까지 배열의 셀을 하나씩 늘려가며 출력한다.

sleep(speed)를 통해서 speed만큼 화면을 멈추어 한 글자씩 출력되도록 한다.

* + 3번 메뉴는 ‘종료되었습니다’라는 문구를 출력한 후 exit(0)을 호출해 정상적으로 프로그램을 종료한다.
* **difficultyMenu()**:
  + 콘솔 화면을 지우고 게임 영역을 출력한 후, 난이도 선택 메뉴를 출력한다.
  + 선택 상자를 출력하고, 사용자가 난이도를 선택할 수 있도록 switchDifficultyMenu() 함수를 호출한다.
* **printDifficultyMenu()**: 난이도 선택 메뉴를 출력한다.
* **switchDifficultyMenu()**:
  + 사용자의 키 입력을 받아 난이도를 선택한다.
  + 방향키로 선택 상자를 이동하고, 엔터키를 누르면 선택된 난이도를 실행한다.
  + 선택된 난이도에 따라 runDifficultyMenu(int num) 함수를 호출한다.
* **runDifficultyMenu(int num)**: 사용자가 선택한 난이도를 실행한다. 1번 난이도는 startEasyGame()을 호출, 2번 난이도는 startNormalGame()을 호출, 3번 난이도는 startHardGame()을 호출하여 게임을 시작한다.

**맵 출력**

* **printMap()**: 이중 for문을 사용하여 배열 map의 모든 셀을 순회하면서 각 셀의 값을 확인하여 해당 위치에 맞는 기호를 출력한다. 플레이어 위치는 빨간색 \* 로, 몬스터 위치는 & 기호로, 보물과 깃발은 F 기호로 출력한다.
* **updateMap(int oldX, int oldY, int newX, int newY, char type)**: 게임 요소의 기존 위치(oldX, oldY)를 빈 공간(SPACE)으로 설정하고 화면에서 지운다(gotoxy와 printf 사용). 새로운 위치(newX, newY)에 게임 요소를 배치하고, 화면에 해당 게임 요소를 출력한다.

**객체 이동**

**플레이어 이동**

방향키 입력을 통해 플레이어를 이동시킨다.

* **movePlayer()**:
  + 키보드 입력이 있는지 확인한다(\_kbhit() 함수 사용).
  + 키보드 입력이 있으면 \_getch() 함수로 키 코드를 읽어온다.
  + 입력된 키 코드에 따라 함수를 호출해 플레이어를 이동시킨다.
    - 위쪽 방향키(72): printPlayer(&player, 0, -1) 호출
    - 아래쪽 방향키(80): printPlayer(&player, 0, 1) 호출
    - 왼쪽 방향키(75): printPlayer(&player, -1, 0) 호출
    - 오른쪽 방향키(77): printPlayer(&player, 1, 0) 호출
  + ESC 키(27)가 입력되면 menu() 함수를 호출해 메인 메뉴로 돌아간다.
* **printPlayer(Pointer\* position, int dx, int dy)**:
  + 플레이어의 새로운 위치를 계산한다.
    - newX = position->x + dx
    - newY = position->y + dy
  + 새로운 위치가 벽(WALL)이 아닌 경우, updateMap() 함수를 호출해 맵을 업데이트한다.
  + 플레이어의 현재 위치를 새로운 위치로 갱신한다.

**장애물 이동**

* **moveMonsters()**:
  + 모든 몬스터(monsterNum 개)를 반복하며 이동을 처리한다.
  + 각 몬스터의 이전 위치와 새로운 위치를 저장하고, moveMonster() 함수를 호출해 몬스터를 이동시킨다.
* **moveMonster(Pointer\* monster)**:
  + 몬스터의 현재 위치를 인자로 받아, 몬스터를 이동시킨다.
  + 랜덤하게 이동 방향을 결정한다 (rand() % 4).
  + 몬스터의 새로운 위치(newX, newY)를 계산한 후, 해당 위치가 벽(WALL), 보물(TREASURE), 이득 깃발(GIFT), 벌칙 깃발(PENALTY)이 아닌 경우에만 이동한다.
  + updateMap() 함수를 호출해 맵을 업데이트하고, 몬스터의 위치를 새로운 좌표로 갱신한다.

**깃발과 장애물 충돌**

**보물 깃발**

* checkTreasure()
  + **보물과의 충돌 검사**: checkCollision(player, treasure)를 호출해 플레이어가 보물 위치에 있는지 확인한다. 보물 위치에 도달했을 경우, 다음 이벤트를 처리한다.
  + **보물 발견 처리**:
    - treasureFound 변수를 1로 설정해 보물을 찾았음을 표시한다.
    - MessageBox 함수를 사용해 "보물을 발견하셨습니다!"라는 메시지를 출력한다.
  + **점수 계산 여부 확인**:
    - MessageBox 함수를 사용해 점수를 계산할지 묻는 메시지 박스를 출력하고, 사용자의 응답을 받는다.
    - 사용자가 "예"를 선택한 경우 (box == IDYES)
      * system("cls")를 호출해 콘솔 화면을 지운다.
      * endsignal 변수를 1로 설정해 게임 종료를 표시한다.
    - 사용자가 "아니오"를 선택한 경우
      * endsignal 변수를 0으로 설정한다.
      * 보물 위치에 !를 출력해 플레이어가 보물에 다시 쉽게 찾을 수 있게 구분하여 표시한다.
      * map 배열의 해당 위치에 TREASURE를 재설정한다.
      * 플레이어의 위치를 맵의 중앙을 기준으로 한 칸 이동시킨다.
        + 플레이어가 맵의 오른쪽 절반에 있으면 왼쪽으로 한 칸 이동.
        + 플레이어가 맵의 왼쪽 절반에 있으면 오른쪽으로 한 칸 이동.
      * drawPlayer() 함수를 호출해 플레이어를 새로운 위치에 다시 출력한다.

**이득 깃발**

* checkGift(): 플레이어가 이득 깃발에 도달했는지 확인하고, 각 깃발의 효과를 적용한다.
  + 플레이어와 선물 깃발의 충돌을 확인한다 (checkCollision 함수 사용).
  + 충돌한 깃발을 제거하고, 깃발의 인덱스에 따라 다른 이벤트를 처리한다:
    - 3개의 깃발: favorableQuestion() 함수 호출.
    - 2개의 깃발: 점수 300점 증가, eventPrint() 함수 호출.
    - 마지막 2개의 깃발: revealTreasureDirection(i) 함수 호출.
  + 랜덤 확률에 따라 rand\_store() 함수를 호출한다.
* favorableQuestion(): 플레이어에게 질문과 3개의 답변 옵션을 출력한다. 플레이어는 1, 2, 3 중 하나를 선택할 수 있으며, 선택된 답변에 따라 점수를 증가시킨다. 질문 번호를 증가시키고, 마지막 질문에 도달하면 다시 처음 질문으로 돌아간다. 질문 메뉴를 지운다.
* eventPrint(): 호감도를 300점 증가시킨 후, 호감도가 증가했다는 내용을 출력한다.
* revealTreasureDirection(int i): 플레이어와 보물의 상대적인 위치를 기준으로 위아래 또는 좌우 방향을 알려준다.
* rand\_store(): 랜덤 상점 이벤트를 처리한다. 플레이어에게 상점 메뉴를 출력한다. 플레이어는 1, 2, 3 중 하나를 선택할 수 있으며 선택된 옵션에 따라 게임 시간을 소모하거나 증가시키고, 플레이어와 몬스터의 속도를 조절한다. 상점 메뉴를 지운다.

**벌칙 깃발**

* **checkPenalty()**:
  + 플레이어와 벌칙 깃발의 충돌을 확인한다 (checkCollision 함수 사용).
  + 충돌한 깃발을 제거한다 (penalty[i].x = 0, penalty[i].y = 0)
  + penalty\_func() 함수를 호출한다.
* **penalty\_func()**:
  + 랜덤하게 벌칙을 결정한다 (rand() % 2 사용).
  + penalty\_rand 값에 따라 벌칙을 처리한다:
    - 0: 몬스터의 속도를 2배로 빠르게 (monstertick \*= 0.5).
    - 1: 플레이어의 속도를 1.5배로 느리게 (playertick \*= 1.5).
  + penaltyPrint(int penalty\_rand) 함수를 호출해 벌칙 내용을 화면에 출력한다.
* **penaltyPrint(int penalty\_rand)**: 벌칙 깃발 발견 메시지를 출력한다.

**장애물**

* checkObstacle():
  + monsterNum 개의 몬스터를 순회하며 각 몬스터와 플레이어의 충돌을 확인한다.
  + 충돌 했을 경우
    - 점수 감소: 플레이어가 몬스터와 충돌하면 점수가 100점 감소한다.
      * 현재 점수가 100점 이상인 경우 100점을 감소시킨다.
      * 현재 점수가 100점 미만인 경우 점수를 0으로 설정한다.
    - 플레이어 위치 이동:
      * 플레이어가 맵의 중앙을 기준으로 오른쪽에 위치한 경우, 왼쪽으로 한 칸 이동한다.
      * 플레이어가 맵의 중앙을 기준으로 왼쪽에 위치한 경우, 오른쪽으로 한 칸 이동한다.
      * drawPlayer() 함수를 호출해 플레이어의 새로운 위치를 출력한다.

**난이도 설정**

게임 시작 전 메뉴에서 난이도를 선택할 수 있다. 각 난이도 별로 달성해야 하는 점수와 몬스터 수가 다르다.

* **startEasyGame():** 쉬운 난이도의 게임을 시작한다. level을 1로 초기화하고, monsterNum을 EASY\_MONSTER로 초기화한다.
* **startNormalGame():** 보통 난이도의 게임을 시작한다. level을 2로 초기화하고, monsterNum을 NORMAL\_MONSTER로 초기화한다.
* **startHardGame():** 어려운 난이도의 게임을 시작한다. level을 3로 초기화하고, monsterNum을 HARD\_MONSTER로 초기화한다.
* **levelUp():** 난이도를 올린다.
  + 게임 변수를 초기화하고, 현재 레벨이 3이 아닌 경우 레벨 업 메시지를 출력한다.
  + 현재 레벨이 3인 경우 게임 클리어 메시지를 출력하고, 메인 메뉴로 돌아간다.

**게임 진행**

* **gameStart()**:
  + **초기화**:
    - initBoard(): 게임 보드를 초기화한다.
    - printMap(): 초기화된 보드를 출력한다.
    - start\_time = time(NULL): 게임 시작 시간을 저장한다.
    - player\_std\_time와 monster\_std\_time을 현재 시간(clock())으로 초기화한다.
  + **메인 게임 루프**:
    - 루프 내부에서 계속해서 다음 작업을 수행한다:
      * **printArea():** 게임 영역을 출력한다.
      * 현재 시간과 경과 시간을 계산한다 (player\_duration, monster\_duration).
      * **checkFlag():** 게임 내 충돌 이벤트를 확인한다.
      * **플레이어 이동**:
        + 경과 시간이 playertick 이상일 때 movePlayer()를 호출해 플레이어를 이동시키고, player\_std\_time을 갱신한다.
      * **몬스터 이동**:
        + 경과 시간이 monstertick 이상일 때 moveMonster()를 호출해 몬스터를 이동시키고, monster\_std\_time을 갱신한다.
      * **시간 업데이트**:
        + 매 1초마다 경과 시간과 남은 시간을 계산하고, printProgressBar()를 호출해 프로그래스 바를 업데이트한다.
        + 점수와 현재 레벨을 출력한다.
        + 남은 시간이 0 이하일 경우 endsignal을 -1로 설정해 게임 종료를 표시한다.
      * **게임 종료 확인**:
        + checkGameEnd()를 호출해 게임 종료 조건을 확인하고, 종료 조건이 만족되면 루프를 종료한다.
      * 과도한 반복을 방지하기 위해 Sleep(5)를 호출해 5밀리초 동안 대기한다.
* **printProgressBar(int elapsed\_time, int gametime)**:
  + 경과 시간과 총 시간에 따라 프로그래스 바의 위치를 계산한다 (position).
  + 경과 시간을 MM:SS 형식으로 출력한다.
  + 프로그래스 바를 출력한다:
    - 현재 위치까지 = 기호를 출력하고, 현재 위치에 # 기호를 출력하며, 나머지 부분에는 - 기호를 출력한다.
  + 총 시간을 MM:SS 형식으로 출력한다.
  + 출력된 내용을 즉시 콘솔에 반영한다 (fflush(stdout)).

**게임 종료**

* **checkGameEnd():** 게임 종료 조건을 검사한다.
  + **보물 찾기 확인**:
    - endsignal == 1인 경우 endGame(1)을 호출한다.
    - 1을 반환해 게임이 종료되었음을 나타낸다.
  + **시간 초과 확인**:
    - endsignal == -1인 경우 endGame(0)을 호출한다.
    - 1을 반환해 게임이 종료되었음을 나타낸다.
  + **게임 계속 진행**:
    - 위의 조건에 해당하지 않는 경우 0을 반환해 게임이 계속 진행 중임을 나타낸다.
* **endGame(int result):** 게임 종료 시 결과를 처리한다.
  + **보물 찾기 처리**:
    - result가 1인 경우 calculateScore()를 호출해 점수를 계산한다.
  + **게임 실패 처리**:
    - result가 0인 경우 게임 오버 메시지를 출력한다.
    - printCentered() 함수를 사용해 게임 오버 메시지를 중앙에 출력한다.
    - 7초 동안 메시지를 표시한 후 menu() 함수를 호출해 메인 메뉴로 돌아간다.
* **calculateScore():** 점수를 계산한다.
  + **배열 정의**:
    - requiredScore[]: 각 레벨에서 요구되는 점수를 정의한다.
    - heartColors[]: 점수에 따라 출력될 하트의 색상을 정의한다.
    - heartColorsCode[]: 각 하트 색상의 콘솔 색상 코드를 정의한다.
    - heartDescriptions[]: 각 하트 색상에 대한 설명을 정의한다.
  + **레벨 업 조건 확인**: 현재 점수가 현재 레벨에서 요구되는 점수 이상인지 확인해 levelUpCondition에 저장한다.
  + **하트 색상 및 설명 인덱스 계산**:
    - heartIndex를 점수에 따라 계산한다 (score / 300 - 1).
    - 점수가 0일 경우 heartIndex를 0으로 설정한다.
  + **하트 색상 및 설명 출력**:
    - heartColor를 중앙에 출력한다.
    - 하트 이모티콘을 색상 코드와 함께 중앙에 출력한다:
      * SetColor(heartColorsCode[heartIndex])로 콘솔 색상을 설정한다.
      * printCentered(heartSymbol, 1, 0)으로 하트 심볼을 중앙에 출력한다.
      * SetColor(15)로 콘솔 색상을 기본값으로 재설정한다.
    - heartDescription을 중앙에 출력한다.
    - 5초 동안 메시지를 표시한 후 화면을 초기화한다.
  + **레벨 업 조건 충족 여부에 따른 처리**:
    - 레벨 업 조건을 충족하지 못한 경우
      * 실패 메시지를 중앙에 출력한다.
      * 5초 동안 메시지를 표시한 후 메인 메뉴로 돌아간다.
    - 레벨 업 조건을 충족한 경우
      * levelUp() 함수를 호출해 레벨 업을 처리한다.

**Ⅲ. 토의 사항**

**Case 1: 플레이어가 보물을 찾은 이후 게임을 지속할 때 몬스터에 의해 보물 지워짐**

**문제 정의**

기존 몬스터의 위치는 깃발들과 맵 테두리 위치로 이동할 수 없게 설정되어 있는데, 플레이어가 보물의 위치를 찾은 후에도 게임을 지속하는 경우 몬스터가 이동하며 보물 깃발을 지운다.

**문제 발생 원인**

printPlayer 함수에서 플레이어의 새로운 위치가 배열 map[][]에서 WALL로 지정되어 있지 않은 경우 PLAYER로 지정되도록 하였다. 따라서 플레이어가 보물에 한 번 도달하면 기존 보물의 위치가 배열 map에서 TREAUSRE가 아닌 PLAYER로 바뀌게 되며 몬스터들이 보물 깃발의 위치로 이동할 수 있게 되었다.

**문제 해결 방법**

checkTreausre 함수에서 플레이어가 게임을 계속 진행하겠다고 한 경우, 다시 배열 map에서 현재 위치를 TREASURE로 지정해주어 몬스터들이 보물 깃발의 위치로 이동하지 못하도록 하였다.

**Case 2: Sleep() 함수로 인한 시간 지연 문제**

**문제 정의**

각각 monstertick밀리초 마다 몬스터가 움직이고 playertick밀리초 마다 플레이어가 움직이도록 했는데 의도대로 각자의 tick마다 움직이지 않는다.

**문제 발생 원인**

Sleep(monstertick), Sleep(playertick)를 사용을 했더니 moveplayer() 함수와 movemonster() 함수 둘 다 서로의 Sleep()에 영향을 받았기 때문이다.

**문제 해결 방법**

Sleep()함수를 사용하지 않고 <time.h>헤더의 time\_t와 clock\_t 구조체와 캘린더 시간을 초 단위로 반환하는time()함수와 프로세스가 실행 후 현재 실행 까지 걸린 시간을 밀리 초 단위로 반환하는clock()함수를 사용해서 moveplayer()함수와movemonster()함수가 서로 영향을 받지 않고 각각의 tick마다 실행되게 하였다.

**Case 3: 플레이어가 몬스터와 보물과 만났을 때 효과의 연속적인 처리/맵 경계에서 지정되는 플레이어**

**문제 정의**

플레이어가 몬스터와 만나거나, 플레이어가 보물에 도달한 후 게임을 지속하겠다고 하였을 경우 그 효과(몬스터: 점수 감소, 보물: 보물 발견 알림 창의 출현)가 반복된다.

**문제 발생 원인**

플레이어와 몬스터가 위치를 옮기기 전에 gameStart 함수의 while(1)문 안에서 checkFlag 함수가 계속 수행되기 때문에 효과가 중복되어 나타난다.

**문제 해결 방법**

checkObstacle 함수에서 checkCollision 함수의 반환값이 1인 경우(몬스터와 플레이어가 만난 경우)와 checkTreasure 함수에서 플레이어가 게임을 계속 진행하겠다고 한 경우에 player의 x좌표 값을 1 키웠다. 하지만 이런 경우 맵의 오른쪽 끝에서 몬스터 혹은 보물과 만났을 시에 맵의 오른쪽 경계에 플레이어가 지정되었다. 이를 해결하기 위해 각 checkObstacle 함수와 checkTreasure 함수에서 현재 플레이어의 x좌표가 맵의 너비의 중간보다 오른쪽에 있으면 x좌표 값을 1 빼고, 왼쪽에 있으면 x좌표 값을 1 더하도록 하였다.

**Case 4: 몬스터와 플레이어의 속도 제한**

**문제 정의**

몬스터 속도 / 플레이어 속도 변화 효과가 계속 중첩될 경우 몬스터나 플레이어의 이동 속도가 너무 빨라지거나 느려진다.

**문제 발생 원인**

속도 변화 효과가 한 번 발동되면 게임이 종료될 때까지 지속되기 때문에 속도 변화 효과가 중첩되어 몬스터의 이동 속도가 너무 빨라지거나 플레이어의 이동 속도가 너무 느려지면 원활한 게임 플레이에 지장이 생긴다.

**문제 해결 방법**

몬스터의 이동속도를 느리게, 플레이어의 이동 속도를 빠르게 하는 선택지를 포함하는 rand\_store() 함수를 생성하여 랜덤한 확률로 상점이 열리면 플레이어가 자신의 필요에 따라 적절한 선택지를 선택해 몬스터와 플레이어의 이동속도를 자신에게 유리한 방향으로 조정할 수 있게 했다.

**Ⅳ. 기여도**

* **송은서: 25%** 게임 시나리오, 메인 함수, 메뉴 함수, 게임 함수, 플레이어 이동, 점수 계산, 난이도별 함수, PPT 제작, 보고서 작성
* **홍지아: 25%** 몬스터 이동, 맵 출력, 이득 깃발 제작 및 출력, 게임 설명, 게임 계속 진행, 보고서 작성
* **유지혁: 25%** 몬스터 이동, 게임 틱 계산 함수, 깃발 접촉 함수, 장애물 접촉 함수, 상점, 보고서 작성
* **강민재: 25%** 발표, 게임 종료 함수, 이득깃발 보물위치 힌트, 보고서 작성